A two-step computation of cyclic redundancy code CRC-32 for ATM networks

by R. J. Glaise

In the field of telecommunications, among the numerous cyclic redundancy codes in use, ATM CRC-32 is difficult to compute because it is based on a polynomial of degree 32 that has many more terms (15) than any other CRC polynomial in common use. CRC checking and generation are generally carried out on a per-byte basis, in an attempt to cope with the dramatic increase of the data throughput of higher-speed lines. More calculations are needed to process a new incoming byte of data if the number of terms of the polynomial is large, because more bits of the current intermediate result must be combined to calculate each bit of the next one. This tends to counteract the intrinsic speed advantage of the per-byte computation by requiring that more processing be done at each cycle. This paper describes a method that overrides the intrinsic complexity of the CRC-32 computation. It permits expediting AAL5 messages, which must be segmented into ATM cells, with CRC-32 computed at one end, and reassembled from cells, with CRC-32 checked for data integrity at the destination. The calculation is in two steps: 1) A first division is done on the entire message (until the last cell is received or segmented) by a much simpler polynomial. 2) A second division, on the remainder of the first division by the regular CRC-32 polynomial, is performed only once, in order to obtain the final result.

Introduction

Messages transported through an Asynchronous Transfer Mode (ATM) network must be segmented into short, fixed-length packets, called cells, at the source and reassembled at the destination. Several "adaptation layers" are defined by the ATM standards [1], which specify how the process is carried out. ATM Adaptation Layer 5 (AAL5) is the simplest way of handling the segmentation and reassembly process itself. It also makes better use of the bandwidth, because, unlike the other adaptation layers, it does not require overhead in any of the cells, except for some bytes in the information field of the last one. For these reasons, AAL5 tends to be the preferred method for breaking messages into cells. Because data integrity must be ensured end to end, a standard *cyclic redundancy checking (CRC)* technique is utilized on the *entire* message, and a *frame check sequence (FCS)* is added to the data and transported with the last cell. At the remote end, while reassembly is performed, the message is checked for data integrity. The polynomial chosen by the relevant standards bodies to implement the AAL5 CRC is

0018-8646/97/$5.00 © 1997 IBM

the same 32-degree polynomial used for the fiber distributed data interface (FDDI) [2]:

$$G^*(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1.$$

The large number of terms (15) of this polynomial makes it more difficult to implement than the other CRCs in use, especially when higher-speed lines are considered. Hence, the intent of AAL5 to provide a simple and efficient segmentation and reassembly (SAR) process is somewhat offset by the need to compute and check a complex degree-32 CRC. This paper proposes a method of simplifying the computation in order to facilitate implementation of circuitry for high-speed CRC computation in standard CMOS technology.

State of the art

Contrary to CRC standards (see for example [2]), each of which describes its particular CRC implementation in the form of a linear feedback shift register (LFSR), in which only one bit at a time can be handled, all of the methods for expediting CRC calculation known to the author tend to propose byte-wise processing. (One of the very first papers on this is [3]. Although it was published much later, [4] is more frequently cited in the relevant literature.)

However, we assert that the method of [5], which proposes eliminating the shift-register model and handling the computation directly according to the mathematical basis of CRCs (the algebra of polynomials) permits achieving much better results as far as hardware implementation is concerned. In [5], the per-byte (8-bit byte) computation described requires the equivalent of 114 two-input XORs and an eight-input *XOR operator* to compute the next bit values.

This may still be too much, however, for the very high-speed computation required to process AAL5 ATM connections flowing through OC-12 lines, for instance. One byte is received approximately every 13 ns for an OC-12 line (622×10^6 bits/s) and every 3 ns for an OC-48 line (2.4×10^9 bits/s). Although not all of the traffic of such lines is likely to be AAL5 connections that must be segmented or reassembled, such numbers tend to indicate that the instantaneous processing capability may have to be very high not to degrade the overall performance at a network node.

Simplifying the calculation

All of the methods for computing CRCs known to the author, including the one of [5], have in common the process of dividing the message by the CRC polynomial $G(x)$ chosen by the relevant standardization group to perform the calculation. The new concept developed in this paper consists of carrying out this calculation in two steps:

1. Checking and generating the CRC is done with *another polynomial*, $M(x)$, $M(x) = G(x) \times P(x)$, except at the final step. This polynomial must be a multiple of the CRC polynomial $G(x)$, in order that the remainder of the first division of the message by $M(x)$ be divisible, in turn, by $G(x)$. $P(x)$ is a polynomial of degree as low as possible to keep the degree of $M(x)$ low, while it must be chosen so that the resulting polynomial, $M(x)$, has fewer terms than $G(x)$ in order to simplify the first division. The desirable structure for $M(x)$, to make calculation easier, is further discussed in the next section.
2. The result of the first division, performed on all of the ATM cells constituting the message, is a *fixed-length vector* with degree equal to that of $M(x)$. This vector must then be divided *only once* by $G(x)$ to obtain the final result.

Making *M(x)* "simple"

The polynomial $M(x) = G(x) \times P(x)$ is said to be a simple polynomial if it has fewer terms than $G(x)$. Not any simple polynomial is satisfactory, however, because it is desirable to have the terms well spread out between the maximum (the degree of the polynomial) and minimum (x^0, or 1) terms. If the calculation is carried out on a per-byte basis, the powers of the terms should ideally be at least 8 bits apart, so as not to "overlap" in the calculation.[1] For instance, the following multiple of the CRC-32 polynomial $G(x)$,

$$M(x) = x^{123} + x^{120} + x^{80} + x^{74} + x^{53} + x^{45} + 1$$

(see Footnote 2),

which has only seven terms, is not as good as the following (referred to hereafter as M^{123}),

$$M^{123} = x^{123} + x^{111} + x^{92} + x^{84} + x^{64} + x^{46} + x^{23} + 1$$

(see Footnote 3),

which has, however, one more term. This is because in this second instance, the powers of the terms are all at least 8 bits apart. With this polynomial, the first division can be carried out 8 bits at a time with only *two-input XORs* by the state machine shown in **Figure 1**. The method is the one described in [5], in which calculations are done in the algebra of polynomials modulo $G^*(x)$. Computing is done here modulo M^{123} so that the result of any operation is a vector that is no more than 123 bits long.

[1] Another way of describing how to select $M(x)$ is as follows: Of those polynomials $M(x) = G(x) \times P(x)$ with terms at least 8 apart, choose the one with the fewest terms. If more than one of these exist, select the one with the lowest degree.

[2] $M(x) = G^*(x) \times P(x)$, where $P(x)$ is the polynomial with the following powers of x: 91 88 85 81 78 76 75 72 67 66 65 64 61 58 56 51 48 47 46 40 39 38 37 36 35 34 33 31 30 27 23 18 16 12 11 9 7 5 3 1 0.

[3] $M^{123} = G^*(x) \times P(x)$, where $P(x)$ is the polynomial with the following powers of x: 91 85 82 81 75 73 70 69 66 65 62 61 59 57 55 54 53 52 49 45 41 40 38 37 36 33 32 31 28 27 26 24 18 16 12 11 9 7 5 3 1 0.

To implement the state machine of Figure 1 *requires only 56 two-input XORs and 123 latches*. The other 67 bits of the next intermediate result are simply shifted bits (by eight positions) of the current result. This permits the state machine to operate at a very high rate, so that the calculation can keep up with the very high speed of optical communication lines commonly used nowadays. This speed is achieved at the expense of more bits to process in parallel and the need to store a wider vector (123 bits instead of 32) with the intermediate result of the computation in progress. This is not really a drawback at present, when gate arrays with more than 100 000 gates are commonly available. Hence, the proposed scheme allows one to trade the size of the vector used to manipulate (and store in latches) for processing speed.

Other compromises are possible if wider XORs (three-input and four-input XORs) can be used while the required processing speed is achieved. Two other polynomials to implement the above computation scheme are listed in **Table 1**, along with M^{123} and $G^*(x)$.

Final division

According to the scheme described here, the final division must still be carried out with $G(x)$. Because this second and final division is now applied to a *short, fixed-length vector* (regardless of the length of the initial message), techniques that are not generally practicable with CRCs, because the message can be of any size, may be considered. Among them, the simplest consists of implementing the method always used with error-correcting codes (ECCs) employed to improve memory reliability, thus working on a fixed-size word. A matrix can be devised (the H matrix, in ECC jargon) and implemented in the form of a combinatorial array that performs the final division. The input to the array is the remainder of the first division (for instance, a 123-bit vector if M^{123} is selected), and the output is the 32-bit vector remainder of the division by $G^*(x)$.

Such an array of logic is straightforward to derive from $G(x)$. The method for doing it can be found in [6] and in many other publications that deal with ECCs. For instance, the corresponding H matrix for M^{53} is given in **Table 2**. The 53-bit input vector that is the result of the division by M^{53} (indexed 0 to 52) is applied to the 53-column matrix so as to compute a bit value for each of the 32 rows. The 1s in each matrix row represent the bits of the input vector for which parity must be computed in order to get the 32-bit vector (indexed 0 to 31) that is the result of the division by $G^*(x)$. The column labeled "XOR inputs" in Table 2 indicates how many bits of the input must be combined to compute the bit of the corresponding row. A 13-input XOR is required. The speed of such an array can by no means match the cycle time of the state machine previously described; the equivalent of several cycles is necessary to generate the result. Taking into consideration, however, that this calculation is done only *once at the end*, one realizes that the overall computation is much faster than with traditional methods, even if short messages (down to single-cell messages) are considered.

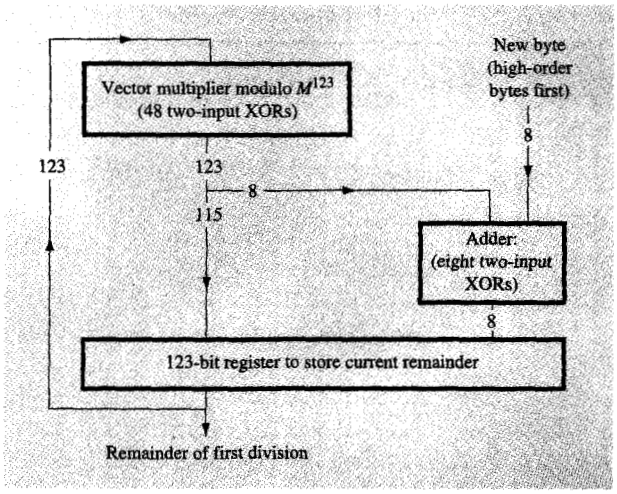

Figure 1

State machine to compute the remainder of the first division by M^{123}. The calculation is done one byte at a time, based on the method described in [5].

Table 1 Polynomials to implement the first division and maximum N-input XOR required.

Polynomials M(x)	*Maximum size of XOR needed*
$M^{123} = x^{123} + x^{111} + x^{92} + x^{84} + x^{64} + x^{46} + x^{23} + 1$	Two-input
$M^{71} = x^{71} + x^{57} + x^{55} + x^{48} + x^{44} + x^{36} + x^{22} + x^{15} + x^{8} + 1$	Three-input
$M^{53} = x^{53} + x^{38} + x^{36} + x^{33} + x^{30} + x^{27} + x^{25} + x^{7} + x^{3} + 1$	Four-input
$G^*(x)$ as implemented in [5]	Eight-input

Method summary

The whole computation scheme is summarized hereafter for M^{123}. As an example, let us assume that the message is one kilobyte long. The state machine cycle time to process one byte can be as low as 10 ns (a 100-MHz state machine) with a two-input XOR. Thus, the 48 bytes of

Table 2 Combinatorial array (H matrix) to perform the final division by M^{53}.

53-bit input vector (remainder of division by M^{53})	XOR inputs
52 ↓ ..... ↓ 0	
.....1...1.11..1.....1.....	6 ← 31
......1...1.11..1.....1.....	6
.......1...1.11..1.....1.....	6
1.......1...1.11..1.....1.....	7
11.......1...1.11..1.....1.....	8
111.......1...1.11..1.....1.....	9
.111.1...1..1....11........1.....	9
1.111.1...1..1....11........1.....	10
11.111.1...1..1....11........1.....	11
.11.1.1.11.1........1.........1.....	9
..11...1..11...1..............1.....	7
...11...1..11...1..............1.....	7
1...11...1..11...1..............1.....	8
.1...11...1..11...1..............1.....	8
1.1...11...1..11...1..............1.....	9
.1.1...11...1..11...1..............1.....	9
1.1.11..1..111.111..................1.....	12
11.1.11..1..111.111..................1.....	13
.11.1.11..1..111.111..................1.....	13
..11.1.11..1..111.111..................1.....	13
1..1111.1..1....11.11..................1.....	12
.1..1.11...1...1.11.1..................1.....	10
..1....111.1...11.11.....................1.....	10
...1....111.1...11.11.....................1.....	10
....11....1.11.1.11.1.....................1.....	10
1.....1..1..11111.11.....................1.....	11
11.....1..1..11111.11.....................1.....	12
111..1..11..1.1.111.1.....................1.....	13
.111.11...1111...111.....................1.....	13
..111.11...1111...111.....................1.....	13
...11..111.1.11....11.....................1.....	11
....1...1.11..1.....1.....................1	7 ← 0

Remainder of final division by $G^*(X)$ (a 32-bit vector)

each ATM cell are calculated in 480 ns (one cell every 700 ns at 622 megabits per second). The complete message, which comprises 21 cells, requires roughly 10 μs plus the final division, which can be done in five cycles or less (i.e., 50 ns maximum). Thus, the final division accounts for only 0.5% of the total calculation time in this first example. For a single-cell message (the worst case), which is processed in 480 ns, the final division accounts for approximately 10% of the total computation time. The two-step scheme described in this paper is summarized in **Figure 2** for M^{123}.

The three polynomials given in Table 1 are the best that the author was able to find in an exhaustive search up to degree 128 of multiples of $G(x)$. (Only those multiples with consecutive terms having exponents differing by 8 or more were retained. Polynomials M^{71} and M^{53} are actually by-products of this search, which was conducted only to find the polynomial of lowest possible degree that permits the first division to be performed per 8-bit byte while requiring only two-input XORs. M^{123} is the result of this search.) The use of a two-input operator guarantees that the state machine is intrinsically the fastest possible. An improvement of the scheme described in this paper could come only from a polynomial of degree less than 123, with fewer terms, which would require less hardware but provide no speed advantage.

Summary and conclusion

This paper describes a method of computing the AAL5 ATM CRC-32 at very high speed. The division is first carried out with a polynomial $M(x)$ different from the one chosen by the standards, so as to ease the calculation while cells are being received or transmitted. When the last cell is processed, a final (slower) single division by the CRC-32 polynomial $G^*(x)$ must be performed to finish the calculation. This scheme, which is possible because $M(x)$ is a simple multiple of $G^*(x)$, permits performing most of the calculation with a very fast state machine that is able to match the very high throughput needed to handle AAL5 connections transported on optical lines, like OC-12 (622 megabits per second), for which one ATM cell must be processed every 700 ns. The new scheme still allows the use of standard submicron CMOS technology, while a higher-performance one would normally be necessary to make the calculation with traditional methods by means of the polynomial $G^*(x)$ alone.

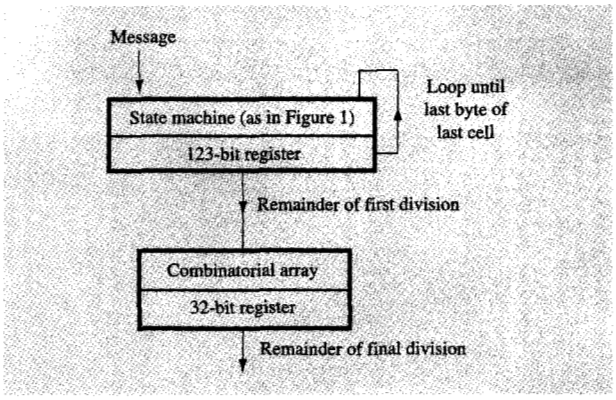

Figure 2

Two-step computation of CRC-32 with M^{123}.

References

1. "Recommendation I.363 on ATM Adaptation Layers," International Telecommunication Union (ITU), Telecommunication Standardization Sector, Geneva, Switzerland, July 1992.
2. "Fiber Distributed Data Interface (FDDI)," *ISO 9314-2*, International Organization for Standardization, Geneva, Switzerland, 1989.
3. A. M. Patel, "A Multi-Channel CRC Register," *Proceedings of the Spring Joint Computer Conference*, 1971, pp. 11–14.
4. A. Perez, "Byte-Wise CRC Calculations," *IEEE Micro*, pp. 40–46 (June 1983).
5. R. J. Glaise and X. Jacquart, "Fast CRC Calculation," *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, Cambridge, MA, 1993, pp. 602–605.
6. W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, 2nd ed., The MIT Press, Cambridge, MA, 1972.

Received May 30, 1996; accepted for publication January 20, 1997

René J. Glaise *IBM Networking Hardware Division, La Gaude Laboratory, 06610 La Gaude, France (rjg@vnet.ibm.com).* Mr. Glaise is a Senior Development Engineer who has long worked on error-correcting codes and storage controllers for the memories of the communication controllers developed at both the La Gaude (France) and Raleigh (North Carolina) laboratories. More recently, he has been involved in the development of an ATM adapter for the IBM Nways switches, for which he has received an IBM Outstanding Technical Achievement Award. Since joining IBM in 1971, just after graduating from the Conservatoire National des Arts et Métiers in Caen, France, he has received six IBM Invention Achievement Awards. His current interests include methods for expediting CRC calculations in data-communication products, along with the design and implementation of powerful search devices.